DEVELOPER

Open Source & Saying "No"

Photo by Roman Synkevych 🇺🇦 / Unsplash

A few weeks ago I was digging into a pretty complex issue that was only affecting less than 1% of users in the field with an application. With a constant mix of emotions working with React Native I appreciate how I can normally go all the way to the source during an investigation.

So I stumble upon a library known as react-native-fs and I learn what I need to, but also find some interesting discoveries.

- It has ~500 open issues.
- It has ~60 open pull requests.
- It has a built in HTTP library for downloading files.
- It has a library built in for uploading files.

So while I'm trying to just research a tiny charset issue with a created file - I learned the feature set of this project is quite large. So in a situation like this I believe the author inherits a good deal of tech debt of things to maintain.

So I get a bit selfish and wonder if the library was specific to just the filesystem and ignored all those extra features - could I anticipate a more stable solution?

From an Open Source maintainer myself I think its extremely important to say "no" to requests/features that bloat your library too much. What I've seen occur many times played out is something like this:

- Talented contributor Z delivers feature X to project Y.
- Project Y accepts it.
- Feature X has an issue in latest release.
- Project Y didn't start that feature so asks contributor Z for some help.
- Contributor Z is not around and the fix is taken in by the project.

So a project shouldn't think about the short term benefit of a new feature, but understand the long term implication of supporting such feature.

I may have accepted a feature to re-add support for 32 bit binaries in Apktool, but I struggled for hours to figure out how to re-add that support when the upstream AOSP project released a new major version of Android.

I did however say "No" to adding Apktool support for the actual Android platform. Since I feared that support of keeping that feature added would be extremely difficult.

So moving back to the filesystem research I started with. I began exploring for a package that might be scoped to just filesystem operations and nothing else.

I found the reversal with rn-fetch-blob an interesting project that started because React Native struggled with properly uploading Blob (binary) contents.

The package exploded with popularity and sure enough added a filesystem implementation to it. So much like we saw with the other filesystem package:

- Forked and continued under a new owner.
- ~450 open issues.
- ~50 open issues.

README.md

File System

File Access

File access APIs were made when developing `v0.5.0`, which helping us write tests, and was not planned to be a part of this module. However, we realized that it's hard to find a great solution to manage cached files, everyone who uses this module may need these APIs for their cases.

Before start using file APIs, we recommend read Differences between File Source first.

File Access APIs

- asset (0.6.2)
- dirs
- createFile
- writeFile (0.6.0)
- appendFile (0.6.0)
- readFile (0.6.0)
- readStream
- hash (0.10.9)
- writeStream
- hash
- unlink
- mkdir
- ls
- mv
- cp
- exists
- isDir
- stat
- lstat
- scanFile (Android only)

See File API for more information

https://github.com/joltup/rn-fetch-blob

So I get selfish again. I'd really wish this library was split between file access and network module. It seems in my research that the smaller a library's feature set is - the more stable it is.

This is probably the Unix philosophy at the end of day and I think I'm pretty aligned with it now. I just want libraries to stay true to a small feature set and do it well.

I'm exhausted from seeing a library abandoned because it expanded too much in features.

I'm exhaused from seeing a library buggy because it expanded too much in features.

I'm exhaused from internal debates rewriting a package and assuming that responsibility to have a smaller isolated feature set.

All in all I'm being much more careful when researching dependencies and libraries before bringing them in.

READ MORE POSTS BY THIS AUTHOR

Connor Tumbleson

OLDER POST

Apktool v2.7.0 Released

Member discussion

0 comments

Start the conversation

Become a member of **Connor Tumbleson** to start commenting.

Sign up now

Already a member? **Sign in**